Enhancing SQL with Set-Comparison Operators

Feng Yu*
Department of Computer Science
and Information Systems
Youngstown State University
Youngstown, OH, USA
Email: fyu {at} ysu.edu

Jing Zhang, Wen-Chi Hou,
Michael Wainer
Akhila Pabbaraju
Department of Computer Science
Southern Illinois University
Carbondale, IL, US

Cheng Luo
Department of Mathematics and
Computer Science
Coppin State University
Baltimore, MD

Abstract—**Queries that involve words like "every", "only", etc., are generally difficult to formulate in SQL. In this paper, we analyze the problems and propose a set of set-comparison operators to enhance the usability of SQL. These operators allow users to formulate conventionally difficult queries in an easier way, greatly enhancing the user-friendliness of SQL.**

Keywords-Query Processing, SQL language, Set Theory

I. INTRODUCTION

First-order logic, which is also called first-order predicate calculus, contains two standard quantifiers, the universal (∀) and existential (∃) quantifiers. SQL, which stands for structured query language, is a data sublanguage proposed in late 1974 [2, 5]. SQL is a tuple relational calculus based on the first-order logic, which has been adopted as an ANSI standard and is used in all relational DBMS products.

While SQL satisfies the necessary requirement of a database query language (i.e., relational completeness [3]), it supports only existential operations directly. The universal quantification is accomplished through using existential operations indirectly. Thus, it is generally awkward to express queries that involve words like "every", "only", etc., which convey the concept of the universal quantifier. Examples of such queries are: (i)" Find items that are sold by every dealer", (ii) "Find the manufacturers who manufacture only items that are sold by every dealer".

Since SQL does not support the "for all" (∀) operation, users have to rephrase the original queries into ones that use negations and existential operations to conform to the SQL syntax. For example, the query "Find items that are supplied by every supplier" is rephrased to a semantically equivalent query - "Find items that no dealer doesn't sell them". The users are often forced to use nested "NOT EXISTS" (or "NOT IN") operators of SQL to express such queries. The translations are not always straightforward and often confusing.

The problems caused by the lack of a "for all (∀)" operator in SQL have also been recognized in [7] [10]. However, it is not straightforward how "for all" can be incorporated into SQL in a consistent and easy-to-use way. In this paper, we address this issue and propose operators that can be used to express traditionally difficult queries in a simple and natural way.

When users pose those queries, they may not realize that they are actually comparing sets. For example, consider the previous queries. In query i), given an item, to see if it is sold by every dealer, we check if the set of dealers that sell this item is the same as the set of all dealers. For query ii), given a manufacture, we check if the set of items he manufactures is a subset of the items that are sold by every dealer. Both queries involve comparing sets of dealers or items to determine if a given item or manufacturer is to be selected into the answers.

We observe that set comparisons are deeply involved in the human thought process in formulating the queries. Therefore, we propose a set of set-comparison operators, which mimic the natural thought process, to replace the "for all" operator. We shall demonstrate how queries can be expressed using the proposed set-comparison operators, as opposed to the traditional ways that use "NOT EXISTS" and/or "NOT IN". As new database applications have gained increasing attention by linguists and logicians, set comparison operators can improve the SQL interface and make queries more self-expressive and easier to understand.

In Section II, we will give more details about the weakness of SQL. In Section III, a set of set-comparison operators are proposed to represent the concept of "for all" in a natural way. In Section IV, we propose to convert queries with the set-comparison operators to traditional SQL to take advantage of existing query optimization techniques. In Section V, the set-comparison operators are compared with other work. Finally, conclusions are given in Section VI.

II. PRELIMINARY

A. Set Queries

The major forms of computation supported by SQL [1, 5] are tuple-level computations and aggregate computations. Tuple-level computations are defined by expressions on variables in calculus expressions or on values of each tuple of a relation in algebraic expressions, whereas aggregate computations apply aggregate functions, such as minimum, maximum, average, count, etc., on sets of tuples. Unfortunately, SQL does not include set computations, which involve comparisons on sets of tuples.

DEALERS

Did	Dealer_name
3405	A America
5301	Oak Express
7803	SK
6409	California Oak

PRODUCE

Mfr	Itemno
Bendwood	292
Green River	155
Richardson Brothers	275
Shin Lee	183

ITEMS

Itemno	Item_name
292	Sunburst Table
155	Trestle Table
275	Butterfly Table
183	Hoopback Chair

SELL

Did	Itemno
3405	155
3405	275
3405	183
3405	292
5301	275
6409	275
6409	292

ORDER

Did	Mfr
3405	Shin Lee
3405	Richardson Brothers
3405	Bendwood
3405	Green River
6409	Bendwood
6409	Richardson Brothers
5301	Bendwood
5301	Green River
7803	Richardson Brothers

Figure 1. Schema of Example Database

B. Example of Hard to Formulate Queries

We show, by example, the shortcomings of using nested "NOT EXISTS" operators in expressing some queries. We shall analyze the problem and propose a solution to these problems.

SQL provides operators to tests emptiness or non-emptiness of a set, i.e., “EXISTS” and “NOT EXISTS”, and membership

between a tuple and a set, i.e., "IN", and "NOT IN". Here are the syntaxes for these operators in SQL:

(NOT) EXISTS (table-expression)

X (NOT) IN (table-expression)

where X is an element.

For simplicity, we shall only mention "EXISTS" and "NOT EXISTS" operators, leaving out the "IN" and "NOT IN" operators, as queries using the latter can always be reformulated by using the former, and vice versa.

Consider a sample database with five relations, as shown in Figure 1. The DEALERS show dealer names and their IDs. Similarly, the ITEMS show the item names and their IDs. The PRODUCE table lists the manufacturers (Mfr) and the items they produce. The SELL table is the list of dealers and the items that they sell. The ORDER table records which dealer orders from which manufacturer.

In the following, we show examples that use regular SQL constructs with “exists” and “not exists” to formulate queries.

Example 2.1 Find items that are sold by every dealer.

The query is translated to: Find items that no dealer doesn't sell them. That is,

```
Select Item_name from ITEMS where not exists (
    select Did from Dealers where not exists (
        select * from SELL where Dealer.Did =
        SELL.Did and SELL.Itemno = ITEMS.Itemno))
```

Example 2.2 Find the manufacturers who produce only items that are sold by every dealer.

The query is translated to: Find manufacturers who do not produce any item not sold by every dealer.

```
Select Mfr from PRODUCE where not exists (
    select Itemno from PRODUCE P
    where P.Mfr = PRODUCE.Mfr and
    exists (
        select Did from DEALERS where
    not exists (
        select * from SELL where DEALERS.Did =
        SELL.Did and SELL.Itemno = P.Itemno
)))
```

Example 2.3 Find dealers who order exactly the same items as dealer SK from the manufacturer Shin Lee.

The query is translated to: Find dealers for whom there does not exist an item that these dealers ordered from Shin Lee but SK did not; and there does not exist an item that SK ordered from Shin Lee, but these dealers did not.

```
Select Dealer_name from DEALERS where
    not exists (
        select Itemno from ORDER O where  O.Dealer_name
        = Dealer_name and O.Mfr = 'Shin Lee'  and not ex-
        ists(
        select * from ORDER R, DEALERS D
        where D.Dealer_name = 'SK' and R.Mfr = 'Shin Lee'
        and R.Itemno =O.Itemno
        ))
    and not exists (
        select Itemno from ORDER D where D.Dealer = 'SK'
        and D.Mfr = 'Shin Lee' and not exists (
        select Itemno from ORDER E where E.Itemno =
        D.Itemno and E.Mfr = 'Shin Lee' and E.Dealer =
        ORDER.Dealer
))
```

Example 2.4 Find manufacturers who produce all the items that no dealer sells.

The query is translated to: Find manufacturers for whom there does not exist an item that no dealer sells and these manufacturers have not produced that item.

```
Select Mfr from PRODUCE where not exists (
    select Itemno from ITEMS where
    not exists (
        select * from SELL where Itemno = ITEMS.Itemno )
    and not exists (
        select * from PRODUCE Q where Q.Itemno =
        PRODUCE.Itemno and Q.Mfr = PRODUCE.Mfr
))
```

C. Causes And Reasons

Researchers [7, 10] recognized that the difficulties of using SQL to express these types of queries lies in the lack of support for the "for all" operators. It is observed that "for all" always involves set comparison on sets of tuples. For example, in Example 2.1, to find items sold by every dealer is to compare the set:{dealers who sell a given item} to another set:{all dealers in the sell table}, as opposed to using "not exists" nested within "not exists" to express the idea of "for all". The operators "not exists" and "exists" are defined to evaluate emptiness of a table; whereas "in" and "not in" are used to examine the membership and non-membership in a set, respectively. None of these operators are designed to compare the relationships between two sets. Consequently, set-comparison operators, which deal with the relationships between sets of tuples, are introduced to improve the SQL's usability.

III. 3. SET-COMPARISON OPERATORS

We propose to incorporate set-comparison operators into SQL to enhance the expressiveness of SQL. First, we present the set-comparison operators and then illustrate their use by example.

A. Set-comparison Operators

Set-comparison operators are operators that compare two sets. They do not manipulate sets but just determine if specified relationships are held between two sets. Here, we design five basic set-comparison operators, which can be used together to cover all possible relationships between two sets [6]. To make these operators easy to use, they are named as closely as possible to our natural language. The five operators are: "has all", "has only", "has some", "has no", and "has other than", as defined in Table I.

Definition 3.1: Let A and B be sets. Let θ denote one of the set-comparison operators. A θ B evaluates to true if the given relationship θ holds between A and B; false, otherwise.

TABLE I. SET-COMPARISON OPERATORS

Set-comparison Operator	Definition	Equivalent Expression
A has all B	$\nexists\, x\, [x \in B \wedge x \notin A]$	$A \supseteq B$
A has only B	$\nexists\, x\, [x \in A \wedge x \notin B]$	$A \subseteq B$
A has some B	$\exists\, x\, [x \in A \wedge x \in B]$	$A \cap B \neq \emptyset$
A has no B	$\nexists\, x\, [x \in A \wedge x \in B]$	$A \cap B = \emptyset$
A has other than B	$\exists\, x\, [x \in A \wedge x \notin B]$	$A - B \neq \emptyset$

In Table 1, all definitions are equivalent to the equivalent expressions based on set theory. Here, we only give the proof of the equivalence between "A has all B" and $\nexists\, x\, [x \in B \wedge x \notin A]$ using set theory. Other proofs are similar, and thus are omitted.

Proof. A has all B means $A \supseteq B$, which is equal to

$$\forall x\, [x \in B \wedge x \in A]$$

Which is equivalent with

$$\nexists\, x\, [x \in B \wedge x \notin A]$$

Thus Definition in Table 1 are well-defined. □

Some examples are as follow:

1) A ⊃ B: A has all and has other than B; A ⊂ B: A has only B and B has other than A;

2) A ≡ B: A has all and has only B;

3) A ≠ B: A has other than B or B has other than A;

4) A ⊄ B: A has other than B (same as A – B ≠ ∅);

5) A – B = ∅: A has only B (same as A ⊆ B)

B. Examples

In this section, we show how to use the set-comparison operators to formulate queries described in Section II.

Example 3.1 Find items that are sold by every dealer.

For each item, we check if the set of dealers that sell this item contains ("has all") the set of all dealers.

```
select Item_name from ITEMS where (
    select Did from SELL where SELL.Itemno =
    ITEMS.Itemno ) has all (
    select distinct Did from SELL
    ))
```

Example 3.2 Find manufacturers who produce only items that are sold by every dealer.

For each manufacturer, we check if the set of items he/she produced contains only ("has only") those items whose dealers contain ("has all") all the dealers.

```
select Mfr from PRODUCE where (
    select Itemno from PRODUCE P where P.Mfr =
    PRODUCE.Mfr) has only (
        select Itemno from SELL where (
            select Dealer from SELL S where S.Itemno =
            SELL.Itemno )
        has all (
                select distinct Dealer from SELL
    )))
```

Example 3.3 Find dealers who order exactly the same items as the dealer SK from manufacturer Shin Lee.

```
select Did from ORDER where (
    select Itemno from ORDER P where P.Dealer =
    ORDER.Dealer and P.Mfr = 'Shin Lee')
has all and has only (
    select Itemno from ORDER Q where Q.dealer = 'SK' and
Q.Mfr = 'Shin Lee' ))
```

The concept "exactly the same" in the query is expressed by the combination of operators "has all and has only". Comparing it with the same query expressed in SQL in Example 2.3, the new expression is much simpler and more intuitive.

Example 3.4 Find manufacturers who produce all the items that no dealer sells.

```
select Mfr from PRODUCE where (
        select Itemno from PRODUCE P
                where P.Mfr = PRODUCE.Mfr)
has all (
        select Itemno from PRODUCE Q
          where Q.Itemno not in (select Itemno from SELL ))
```

Here, we use "has all" to convey the concept of "all" the items in the query. Again, the query is formulated in a more natural and easier way than before. Notice that we also use the "not in" operator in the query. In fact, each operator, including "in", "not in", "exists" and "not exists", has its own advantages in solving problems. For example, "in" and "not in" are natural for evaluating the membership of an element; "exists" and "not exists" are good for judging the emptiness of a table; set-comparison operators are ideal for checking the relationship between two sets. The proposed set-comparison operators should combine with other operators to produce the most natural expressions for queries.

An alternate solution is given in the following. Users can choose the way which seems most intuitive to them to formulate the queries.

```
select Mfr from PRODUCE where (
    select Itemno from PRODUCE P where P.Mfr =
    PRODUCE.Mfr )
has all (
    select Itemno from PRODUCE Q where (
          select Itemno from SELL
    )
    has no {Q.Itemno}
)
```

IV. CONVERTING SET-COMPARISON OPERATORS TO STANDARD SQL

While these set-comparsion operators can be easily implemented, we believe it would be most beneficial to the database system if these operators are converted into standard SQL constructs, with EXISTS, NOT EXISTS, IN, and NOT IN, to take advantage of the existing query optimizers to optimize the executions. In the following, we discuss how to convert these operators.

A. Has All

Let X and Y be two sets of relations in the query, and x_i's and y_i's are attributes of X and Y, respectively. Let condX and condY be conditions specified on X and Y, respectively.

select x_i's from X where condX
has all
select y_i's from Y where condY

⇩

not exists (select y_i's from Y where condY and
not exists (select x_i's from X where
condX and X. x_i's=Y. y_i's));

Examples 2.1 and 3.1 illustrated this conversion.

Proof: Let A be the set of tuples in "select x_i's from X where condX", and B "select y_i's from Y where condY". The first part of the SQL query after the conversion:

"not exists (select y_i's from Y where condY and …)"

is equivalent to $\nexists t\ [t \in B \wedge \ldots]$.

Given a tuple $y \in B$, the second part of the SQL query after the conversion:

"not exists (select x_i's from X where condX and X. x_i's =Y. y_i's)" is equivalent to $t \notin A$.

Thus, the entire statement is equivalent to $\nexists t\ [t \in B \wedge t \notin A]$, which is exactly the definition (as shown in Table 1) of the operator "A has all B". □

Readers can also easily verify the correctness of this conversion by checking the meaning of these two expressions.

B. *Has only*

select x_i's from X where condX
has only
select y_i's from Y where condY

⇩

not exists (select x_i's from X where condX and
not exists (select y_i's from Y where
condY and X. x_i's=Y. y_i's))

Examples 2.2 and 3.2 illustrated the conversion.

Note: All proofs of the correctness of the translations are similar with the proof of A, and thus are omitted from here on.

C. *Has some*

select x_i's from X where condX
has some
select y_i's from Y where condY

⇩

exists (select x_i's from X where condX and
exists (select y_i's from Y where condY
and X. x_i's=Y. y_i's))

One needs not use this operator if he is comfortable with using nested "EXISTS".

D. *Has No*

select x_i's from X where consX
has no
select y_i's from Y where condY

⇩

not exists (select y_i's from Y where condY and
exists (select x_i's from X where condX
and X. x_i's=Y. y_i's))

"Has no" can be used in queries like "Find dealers who do not sell any items in common with dealer SK.

E. *Has other than*

select x_i's from X where condX
has other than
select y_i's from Y where condY

⇩

exists (select x_i's from X where condX and x_i's
not in (select y_i's from Y where condY))

"Has other than" can be used for queries like "Find dealers who sell some different items than SK".

V. Comparisons with Other Work

There has been earlier work [4, 7, 8, 10] related to set queries in the previous two decades. In this section, we describe the two works [4, 7] which are most closely related to ours.

A. *Relational Calculus with Set Operators (RC/S)*

It was pointed out in [10] that the universal quantification (i.e., for all (∀)) and/or negation are difficult to use, and thus none of the current query languages support the universal quantifier, and few support negations.

Wang [10] proposed to incorporate the set comparison and set manipulation operators into relational calculus, called RC/S, and used a graphical language QBE, not SQL, to illustrate the idea. The set comparison operators, θ, defined in RC/S are: $\theta \in \{\subset, \subseteq, \equiv\}$.

Certainly, these operators can be represented directly or indirectly by our operators. While these operators may be used to formulate some queries easily, it may also require more effort for others. Consider the following two queries:

1) S_1 has no element contained in S_2

2) S_1 has some different elements from those contained in S_2.

These two queries can be directly formulated using our "has no" and "has other than". But, in RC/S, a third set S_3, to be described in the following, has to be derived so that S_1 can be compared to S_2.

For 1), let $S_3 = S_1 - S_2$. If $S_3 \equiv S_1$, then the relationship- S_1 has no element contained in S_2 - holds.

For 2), let $S_3 = S_1 - S_2$. If $S_3 \neq \varnothing$, then the relationship - S_1 has some different elements from S_2 - holds.

It is noted that RC/S is designed for relational calculus, not for SQL.

B. Generalized Quantifiers

Hsu, et. al. [6] attempted to adapt linguistic concepts to improve SQL syntax. They try to support basic English constructs like determiners, noun phrases, verbs, and especially quantifiers in the new SQL constructs so that programmers can be relieved from the burden of working with convoluted queries.

There are many natural language phrases, called determiners, that can be used to quantify associated noun phrases. These noun phrases are called generalized quantifiers. A generalized quantifier is a particular kind of operator on sets. Consider the English sentence "Dealers sell at least one item". In this sentence, "at least one" acts as a determiner. "at least one item" is called a quantifier as it is an operator in that it can take a table expression and yield a truth value.

Generalized quantifiers establish a good theory, but there is still much more to be done. An important issue is that there are uncountable natural language phrases that can act as determiners and can form quantifiers. So, how many of them and which of them should be transformed into SQL operators still remain to be answered.

El-Sharkawi's [4] and Li's [8] works are remotely related to ours. Unlike our work, trying to extend the expressive powers of SQL, El-Sharkawi [4] discussed how to efficiently implement existing set comparison operators in SQL, such as In, NOT IN, =, >, …etc. Li, et al. [8] proposed to form sets using GROUP BY operators, and compared them with dynamically forms sets, e.g., {1, 2} {"a", "b"}, etc. in data warehousing and OLAP environments.

VI. Conclusions and Future Work

In this paper, we discussed the difficulties in expressing queries that involve the concepts of comparing sets of tuples in SQL. We proposed to incorporate set-comparison operators into SQL and showed that with these operators, previously difficult queries can be easily formulated. We have also shown that the proposed set-comparison operators are sufficient to cover all possible relationships between two sets.

To take advantage of the existing query optimizers, we proposed to convert queries expressed with the proposed set operators into standard SQL queries. Such conversions are shown to be very simple, which makes the incorporating of these operators into SQL a real possibility.

References

[1] P. Atzeni and V. DeAntonellis, Relational database theory, The Benjamin Cummings, 1993.

[2] D. Chamberlin, R. Boyce, SEQUEL: A Structured English Query Language, Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control, 249–64, 1974.

[3] E. Codd, Relational completeness of data base sublanguages, Database Systems, 1972.

[4] M.. El-Sharkawi: Efficient Processing of Set SQL Queries. Future Databases, 355-361, 1992.

[5] C. Date and H. Darwen, A Guide to the SQL standard. Addison-wesley, 1993.

[6] R. Grimaldi, Discrete and combinatorial mathematics: an applied introduction. Addison-Wesley, 1989.

[7] P. Hsu and D. Parker, Improving SQL with generalized quantifiers. IEEE, Transaction on Software Engineering, 298-305, 1995.

[8] C. Li, B. He, N. Yan, M.Safiullah, Set Predicates in SQL: Enabling Set-Level Comparisons for Dynamically Formed Groups. IEEE Trans. Knowl. Data Eng. 26(2):438-452, 2014.

[9] S. Russell and P. Norvig, Artificial Intelligence: a modern approach. Prentice-Hall, 1995

[10] G. Özsoyoglu, H. Wang: A Relational Calculus with Set Operators, Its Safety and Equivalent Graphical Languages, IEEE Trans. Software Eng. 15(9): 1038-1052, 1989